OpenFOAM and Third Party Structural Solver for Fluid–Structure Interaction Simulations

Robert L. Campbell
rlc138@arl.psu.edu

Brent A. Craven
bac207@arl.psu.edu

Fluids and Structural Mechanics Office
Applied Research Laboratory
The Pennsylvania State University

6th OpenFOAM Workshop
Penn State University
University Park, Pa

13-17 June 2011
Objective

- Demonstrate a method to include a third-party, structural solver of choice for FSI simulations (quasi-steady)

- Why important? Structural analysts seem to have strong ties to their structural software and want to carry them forward for FSI simulations
● Brief Introduction to Fluid–Structure Interaction
  – Monolithic
  – Partitioned
    • Loose coupling
    • Tight coupling

● Solver Implementation
  – Flow solver
  – Structure solver
  – Solver interface

● Example Problem
  – Setup
  – Solution
Fluid–Structure Interaction (FSI) modeling is a type of multi-physics modeling that combines physical models of fluids and structures.

FSI, in this context, implies two-way coupling.

Two approaches to FSI modeling:

- **Monolithic**
  - Governing equations for both the fluid and solid cast in terms of the same primitive variables.
  - Single discretization scheme applied to entire domain.

- **Partitioned**
  - Fluid and solid domains modeled separately.
  - Separate discretizations of each domain.
  - Stress and displacement communication across the domain interface.
  - Two coupling schemes: Loose and Tight.
FSI Loose Coupling

- Loose coupling approach does not check for convergence at each time step:

\[ u^p \approx u \]

→ This is the source of the artificial added mass (fluid forces computed based on predicted instead of actual displacements)

→ Decreasing time step size does not necessarily improve stability

→ Instabilities most common when fluid/solid density ratio large (≥ 1)
FSI Tight Coupling

- Subiterations each time step to ensure fluid and structure interaction is converged
  - Fixed-point iteration shown here with under-relaxation

\[
\begin{align*}
\mathbf{u}_i &= \omega_i \mathbf{u}_i - (1 - \omega_i) \mathbf{u}_{i-1} \\
\mathbf{u}\_i &= \mathbf{u}_{i+1}
\end{align*}
\]
Solver Implementation

- Using partitioned approach: OpenFOAM and third-party structural solver (e.g., Abaqus, Nastran, etc., but only a skeleton provided here for demonstration purposes)
  - OpenFOAM solves the flow field
  - Interface stresses applied to structural boundary
    - Sample solver requires matching interface meshes
  - Interface boundary displacements determined from resulting structural deformation
  - Fluid mesh deformed

- Partitioned approach requires fluid mesh motion to accommodate structural deformation
  - Solution-dependent motion (cannot be prescribed a-priori)
  - Use OpenFOAM’s dynamicFvMesh capability (refer to Dr. Jasak’s “Dynamic Mesh Handling in OpenFOAM”)

- Requirement of matching interface meshes can be relieved with addition of interpolation scheme
  - OpenFOAM’s GGI
  - Custom interpolation
Fluid Mesh Motion

- FSI with partitioned solver requires motion of fluid mesh to accommodate structural deformation
- Use Laplace face decomposition approach
  - Laplace equation solved using boundary velocity (diffusion equation with variable diffusivity)
  - Must specify velocity of interface vertices
    - Beware – tet decomposition of poly patch adds vertices
    - Can use tet poly patch interpolation

- Key to Success: Getting hook into motionU for solution-dependent specification of boundary motion ⇒ Everything else for mesh motion is automatic!
Start with SimpleFOAM
- Rearrange solver to iterate on flow solution within an outer time loop
- Originally constructed for viscoelastic structures (slow deformations with time \(\Rightarrow\) quasi-steady simulation)

```cpp
while (runTime.loop())
{
  ...
  // Pressure-velocity SIMPLE corrector
  {
    #include "UEqn.H"
    #include "pEqn.H"
  }
  ...
}
```
Start with SimpleFOAM

- Rearrange solver to iterate on flow solution within an outer time loop
- Originally constructed for viscoelastic structures (slow deformations with time $\Rightarrow$ quasi-steady simulation)

```cpp
while (runTime.loop())
{
    ...
    do
    {
    ...
        // Pressure-velocity SIMPLE corrector
        {
            #include "UEqn.H"
            #include "pEqn.H"
        }
        if(fluidConverged)
        {
            solveStructureAndmoveFluidMesh();
            iterCnt++;
        }
    } while(iterCnt < numStrIter)
    ...
}
Solver Overview

- **Create a structure class (example provided)**
  - Can be a wrapper around commercial software (e.g., Abaqus)
  - Can be open source software (e.g., Calculix)
  - Key member functions:
    - applySurfaceStress
    - solve
    - getDisplacements

- **Create an fsiInterface class (example provided)**
  - Exchange stress and displacement information with the structure object
  - Key member functions:
    - transferBoundaryStressToStructure
    - solveStructure
    - moveFluidMesh
  - Example interface requires matching interface meshes
Test Case

Uniform Inflow (ramped on)

Flexible Structure

1 m

1.5 m

0.6 m

0.1 m

3 m
- **Fluid mesh**: use blockMesh utility to create 2D mesh

  ![Diagram of fluid mesh with labels for inlet, outlet, top, and bottom]

- **Structure mesh**: fictitious structural solver used for demonstration – only the interface face and node locations are specified via input file
  - If you change discretization in blockMeshDict, then must also change structure.inp file
A skeleton structural class is provided
- Reads element interface information from a file
- Distorts the geometry based on the computed maximum interface pressure
  - \( d_{xi} = (y_i/H)^2 \times 0.5 \times P_{max} / 50. \)
  - 0.5 is \( \approx \) maximum tip displacement for baseline case (in meters)
  - \( d_{xi} \) is the x-component of displacement of node i
  - \( y_i \) is the y-coordinate of node i
  - H is the height of the structure
  - \( P_{max} \) is the maximum interface pressure
  - 50 is (approximately) the maximum interface pressure for the default case

This class can be augmented to include a structural solver of your choosing
File Structure

- All files same as CFD case (with dynamic mesh), but with the following additions
  - constant/couplingProperties (new file)
    - Fluid patch name that interfaces with structure
    - Structure folder and file name of structure definition
    - Moving region of mesh
  - structure/structure.inp (new file)
    - structure boundary definition as required by the structure object
  - system/controlDict (new entries)
    - maxConvIter: limit number of fluid iterations per solution step
    - fluidConvergenceCrit: fluid convergence criterion
    - maxStructureIter: number of fluid/structure subiterations per time step
Test Case Results

- Deformations computed using default values of test case
- Exercise:
  - Change maximum flow speed (via inletVelocityRamp) from 0.1 to 0.2 and re-compute response

Maximum $U_x = 0.1$ m/s
Test Case Results

- Deformations computed using default values of test case
- Exercise:
  - Change maximum flow speed (via inletVelocityRamp) from 0.1 to 0.2 and re-compute response

Maximum $U_x = 0.2$ m/s
Summary

- **Skeleton classes provided for a partitioned FSI solver using**
  - OpenFOAM for flow solver and fluid mesh motion
  - Third-party structural solver

- **Interface class developed to perform stress and displacement communication between solvers (skeleton class provided)**
  - Requires matching interface meshes
  - Member functions allow sub-iterations for tight coupling

- **Skeleton structure class provided**
  - Can be evolved into a wrapper function for structural solver of choice
Questions/Comments?